

SIGNAL-01

Avoid using signals or at least keep them short to avoid preemptive interference and reentrancy issues.

Sean Barnum, Cigital, Inc. [vita¹]

Copyright © 2007 Cigital, Inc.

2007-04-16

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 6087 bytes

Attack Category	<ul style="list-style-type: none">• Denial of Service• Privilege Exploitation	
Vulnerability Category	<ul style="list-style-type: none">• Process management• Threading and synchronization problem	
Software Context	<ul style="list-style-type: none">• Process Management• Debug API	
Location	<ul style="list-style-type: none">• signal.h	
Description	<p>The signal() system call installs a new signal handler for the signal with number signum. The signal handler is set to sighandler, which may be a user-specified function or either SIG_IGN or SIG_DFL. Upon arrival of a signal with number signum the following happens. If the corresponding handler is set to SIG_IGN, then the signal is ignored. If the handler is set to SIG_DFL, then the default action associated with the signal (see signal(7)) occurs. Finally, if the handler is set to a function sighandler, then (1) either the handler is reset to SIG_DFL or an implementation-dependent blocking of the signal is performed and (2) sighandler is called with argument signum.</p> <p>Using a signal handler function for a signal is called "catching the signal." The signals SIGKILL and SIGSTOP cannot be caught or ignored.</p> <p>Because signal calls are preemptive and not necessarily reentrant, there exists the possibility of one signal being called, its handler beginning to execute and then during execution, a second signal being called. In this case, the first signal handler's execution is stopped and the second signal handler executes but when it finishes, It does not necessarily reenter the first signal handler to finish its execution. This leaves the system in an unknown state.</p>	
APIs	Function Name	Comments
	signal	

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)

Method of Attack	An attacker could rapidly call various signals over and over causing preemptive and reentrant conflicts between signal handlers leaving the system in a potentially unstable state.		
Exception Criteria			
Solutions	Solution Applicability	Solution Description	Solution Efficacy
	Generally applicable.	Minimize use of signals. They are nonportable and their behavior varies greatly from system to system.	Effective
	Generally applicable.	Minimize signal handler functionality to reduce the potential window for overlap between signal calls. Keep signal handlers small and straightforward. A good strategy is to use the signal handler to set a flag of some sort that another part of the program will look for and act on. Do not allow signals to run at privileged level.	Makes exploitation more difficult but does not resolve the vulnerability completely.
	Generally applicable.	Encase signal handlers in a self-checking wrapper to ensure that a given handler is nonselfreentrant (i.e., that calling signal1 while they signal1 handler is executing	Effective at eliminating risk of selfreentrant signal conflicts but does not effect potential intersignal conflicts.

		does not restart signal handler but ignores any signal calls until the signal handler finishes execution).	
	Generally applicable.	Warn on instances of signal. If possible, trace which function is being registered as the signal handler. If the function is overly large or complex, increase the level of severity of the warning.	Effective at making developer aware of potential issues.
Signature Details	<pre>typedef void (*sighandler_t)(int); sighandler_t signal(int signum, sighandler_t handler);</pre>		
Examples of Incorrect Code			
Examples of Corrected Code			
Source References	<ul style="list-style-type: none"> • Rough Auditing Tool for Security (RATS)² • signal(2) - Linux man page³. • Schilling, Jonathan. C++ and Signal Handling⁴. 		
Recommended Resource			
Discriminant Set	Operating System	<ul style="list-style-type: none"> • UNIX (All) 	
	Languages	<ul style="list-style-type: none"> • C • C++ 	

Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about “Fair Use,” contact Cigital at copyright@cigital.com¹.

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. <mailto:copyright@cigital.com>